Loop-Fusion Assignment

Amanjot Singh matr. 152792
Leonardo Temperanza matr. 152831

Implementazione

Poter poter applicare questa trasformazione, è necessario che due loop soddisfano le seguenti condizioni (supponendo che un loop A precede un altro loop B nel CFG):

1. I loop devono essere adiacenti, il che significa che l'uscita del loop A coincide con il preheader del loop B. Non ci devono essere istruzioni tra i due loop, quindi il preheader deve soltanto contenere un'istruzione, ovvero il salto incondizionato verso l'header del loop B.
2. Devono eseguire lo stesso numero di iterazioni, e i "bounds" del loop devono coincidere.
3. Devono essere equivalenti dal punto di vista del control flow
4. Non possono esserci "negative distance dependencies", ovvero dove il loop B (che succede il loop A) dipende da un valore calcolato in un'iterazione futura di loop B. Per semplicità, questo punto non viene implementato.

I vantaggi principali di fondere assieme i loop sono il riutilizzo di dati che verrebbero altrimenti ricalcolati più volte, ma soprattutto i numerosi vantaggi per quanto riguarda le possibilità di parallelizzazione, utilizzo migliore della cache, minimizzazione dell'utilizzo della banda di memoria. La prima fase è alquanto auto-esplicativa, mentre la seconda necessita di informazioni sui "bounds" dei loop, che vengono fornite solo dopo il passo di "loop-rotate".

La forma "ruotata" dei loop può essere definita come una forma più canonica della forma canonica (data dal passo loop-simplify), che trasforma ciascun loop in un if-statement con un do-while al suo interno (ragionando ad alto livello). Il loop risultante è comunque un loop in forma loop-simplify con l'aggiunta di un guard. Questa forma viene usata da molti passi LLVM, ad esempio la LICM (Loop Invariant Code Motion). Lo spostamento in sé avviene rimpiazzando gli usi dell'induction variable del loop B con quella del loop A, dopodichè si può modificare il CFG in modo tale che venga eliminato il Latch del loop A e il Preheader del loop B. Per fare questo è necessario sostituire un'induction variable con un'altra, e modificare i riferimenti i blocchi da eliminare in modo opportuno.

Misurazioni

L'obiettivo di questa porzione è quello di misurare le differenze di performance, in particolare quelle causate da un migliore utilizzo della gerarchia di memoria.

Il primo passo è quello di creare file oggetto per la versione normale e quella ottimizzata (usando solamente l'ottimizzazione di loop-fusion).

Per la versione priva di ottimizzazione: si eseguono i seguente comandi:

```
clang -O0 -Xclang -disable-O0-optnone -emit-llvm -c Loop.c
```

```
opt -passes=mem2reg,loop-simplify,loop-rotate Loop.bc
```

Il primo comando lancia una compilazione in modalità -O0, ovvero senza ottimizzazioni, e tramite l'opzione -Xclang specifica al compilatore l'opzione -disable-O0-optnone, per permettere di effettuare ottimizzazioni sul file prodotto (come comportamento predefinito in modalità -O0 vengono marcate le funzioni come "optnone", impedendo future ottimizzazioni). Tale comando genera quindi una rappresentazione intermedia (per via di -emit-llvm). Si effettuano le ottimizzazioni/analisi propedeutiche alla Loop-Fusion, ovvero mem2reg, loop-simplify, loop-rotate.

Un primo test che si può fare per misurare la differenza di cache-miss è il seguente. Con la utility perf di linux è possibile visualizzare il numero di cache miss di un qualsiasi processo che si vuole lanciare. È stato utilizzato il seguente comando:

```
perf stat -B -e cache-references,cache-
misses,cycles,instructions,branches ./a.out 400000000 1
```

Il codice presente nel file oggetto Loop.o viene eseguito per un array di 400,000,000 elementi (int, 4-byte). Si può immediatamente notare la differenza in cache-miss; la prima parte è relativa al file oggetto non ottimizzato, a differenza della seconda parte. Tutte le misurazioni sono state effettuate con un i5-9600KF (5.00GHz)

```
Performance counter stats for './a.out 400000000 1':

       127.166.277      cache-references
        97.352.356      cache-misses              #   76,555 % of all cache refs
     7.326.073.998      cycles
     8.565.747.280      instructions              #    1,17  insn per cycle
     1.459.548.389      branches

       1,466209093 seconds time elapsed

       0,576835000 seconds user
       0,889287000 seconds sys


Performance counter stats for './a.out 400000000 1':

        86.197.973      cache-references
        63.991.339      cache-misses              #   74,238 % of all cache refs
     6.946.733.394      cycles
     6.966.946.443      instructions              #    1,00  insn per cycle
     1.060.734.348      branches
```

```
1,391378487 seconds time elapsed

0,529249000 seconds user
0,862034000 seconds sys
```

Il secondo test misura invece il tempo di esecuzione delle due alternative. È stato realizzato un test automatizzato che compila tutte le misurazioni di tempo (medie su 100 tentativi) al variare della dimensione dell'array su cui iterare, e lo step di iterazione. All'aumentare dello step di iterazione si aumenta anche proporzionalmente la dimensione dell'array in modo che il numero di iterazioni rimanga costante. Occorre tenere in considerazione che le misurazioni corrispondenti a step elevati potrebbero risultare più basse a causa dell'hardware-prefetching. Il processore analizza il pattern di accesso e cerca di caricare sulla cache i dati di cui si avrà bisogno (ipoteticamente). Un altro fattore da considerare è la banda di memoria, ovvero la quantità di dati che possono essere letti/scritti in memoria principale, espressa in byte/secondo.

I dati sono compilati e graficati nel file "results.xls".